

PROCESADORES DE LENGUAJES

Práctica Obligatoria

Entrega final

GRADO EN INGENIERÍA INFORMÁTICA
ETSII



Universidad
Rey Juan Carlos

Escuela Técnica Superior
Ingeniería Informática

Grupo 30

Gonzalo de la Cruz Cepeda

Álvaro Brea Arjona

Contenido

Introducción:	3
Gestión de errores:	3
Recuperación por escaneo de los siguientes tokens:	4
Recuperación por tokens equivocados:	4
Recuperación por errores en subreglas:.....	4
Recuperación de error Fallo-Seguro:	4
Trabajo realizado:.....	5
Traducción dirigida por sintaxis:.....	6
Nivel Aprobado:.....	6
Nivel Opcional:	6
Casos de prueba:	6
Caso1B:.....	8
Caso2A:.....	9
Caso2B:.....	9
Caso3A:.....	10
Caso3B:.....	11
Caso4A:.....	12
Caso4B:.....	13
Caso5:.....	14
Anexo:.....	15
Cabeceras:	15
Siguietes:	15
Directores:	16

Introducción:

La práctica consiste en implementación de un visualizador de código fuente para un lenguaje de programación.

Esto se llevará a cabo partiendo de un fichero de texto de entrada con el código, y terminará con un fichero HTML en el que aparecerá dicho código, mostrado de forma correcta, añadiendo tabulaciones y haciendo uso de las reglas de diseño en la programación.

Para llevar a cabo dicha tarea, haremos uso de los lenguajes ANTLR v4, Java y HTML.

Gestión de errores:

Para este apartado, hemos añadido una nueva clase llamada `errorListener.java` para sustituir el `errorListener()` que actuaba por defecto. En él hemos seguido los pasos explicados en el libro de ANTLR recomendado por la asignatura. Hemos añadido un par de métodos, uno que nos señale la ubicación del error, otro que nos traduzca los cuatro tipos de errores mencionados en el libro.

En cuanto a la gestión de errores a la hora de generar el HTML, hemos hecho que cuando el código introducido, tenga algún error, no se generará ni código HTML, ni el propio fichero “.html”, a excepción del caso 5.

```
< Error numero 1 >
Partes afectadas en el error: [r, program, part, restpart, blk, sentlist, sent]

línea 3:28 en [@12,66:66=';',<11>,3:28]: Entrada no esperada ';' se esperaba 'entonces'
Error sintactico: Puede que este error se deba a que un token fue añadido donde no debía.
    bifurcacion (a == 5);
                ^

< Error numero 2 >
Partes afectadas en el error: [r, program, part, restpart, blk, sentlist, sent, blk, sentlist, sentlist_r, sent, blk]

línea 7:22 en [@22,158:162='hasta',<19>,7:22]: No coincide la entrada 'hasta' se esperaba 'inicio'
Error sintactico: Puede que este error se deba a que varios tokens fueron añadidos donde no debían.
    bucle hasta(cierto)
            ^^^^^

< Error numero 3 >
Partes afectadas en el error: [r, program, program_f, program, part, restpart, blk, sentlist, sent]

línea 21:25 en [@65,494:494='i',<40>,21:25]: falta ';' en 'i'
Error sintactico: Puede que este error se deba a que falta un cierto token en la ubicación antes mencionada.
    buclepara (i = 0 i <= var6; j = -2)
                ^
```

Recuperación por escaneo de los siguientes tokens:

Cuando el parser se encuentra con un error del que no hay recuperación posible de otra manera, empieza a ignorar tokens hasta que reconozca un estado conocido. Ese estado conocido es básicamente un token desde el que puede volver a continuar analizando.

A este método como el “modo pánico”, pero en este caso funciona realmente bien.

El parser, cuando no encuentra la entrada necesaria para el analizador sintáctico, empieza a ignorar tokens hasta llegar a un estado conocido desde el que se pueda recuperar.

Recuperación por tokens equivocados:

Una de las operaciones más utilizadas por el parser, es el de hacer un “match” o emparejar tokens con la gramática. Si el token que está siendo reconocido, se notifica a los *ErrorListeners()* y se intentará resincronizarlo.

Para resincronizarlo, hay tres formas posibles: Puede borrar el token, puede añadir uno o puede lanzar una excepción para activar el mecanismo de “sync-and-return”.

Recuperación por errores en subreglas:

Esta estrategia de recuperación se utiliza en situaciones en las que se da un error dentro de un bucle. ANTLR intenta realizar la eliminación de un token utilizada en la estrategia anterior, y si no funciona, y el error se produce después de haber reconocido una iteración correcta del bucle, ANTLR ignora tokens hasta que encuentra algo que encaje con el contenido del bucle, lo que va justo después, o el conjunto de resincronización.

Recuperación de error Fallo-Seguro:

Por último, los parsers generados por ANTLR tienen una función llamada “fail-safe”, que garantiza que la recuperación de error termine si entra en un bucle, forzando a consumir un token si llega a la misma posición del parser con la misma entrada.

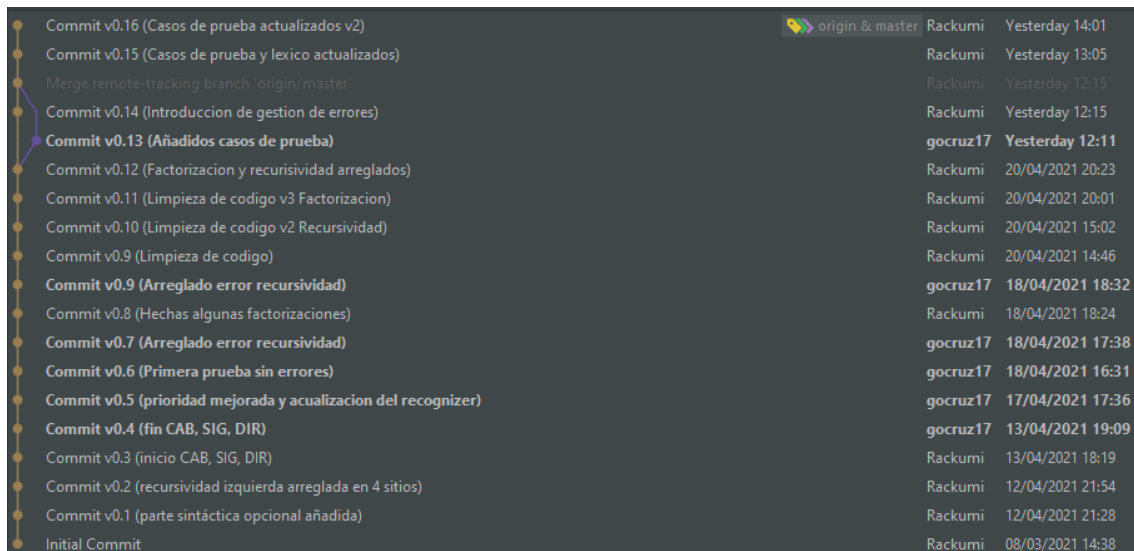
Trabajo realizado:

Para facilitar el trabajo en grupo y más de forma remota, hemos hecho uso de Github, y a continuación se encuentra una imagen del control de versiones del proyecto, en el que se han ido añadiendo las distintas partes del proyecto. Dicha imagen es antigua ya que, si pusiésemos una imagen de la actual, serían más de 60 commits. En primer lugar, desarrollamos el analizador léxico para que nuestra gramática sea capaz de reconocer los identificadores, las constantes etc.

A continuación, añadimos la parte del analizador sintáctico que nos fue facilitado en el enunciado de la práctica. Empezando por la parte obligatoria seguida de la parte opcional. Este fragmento de código no cumplía las propiedades necesarias como para ser una gramática LL(1) ya que tenía recursividad por la izquierda y también requería de factorización. Así que con ayuda de las cabeceras y los directores que calculamos y que se incluyen más adelante en el anexo, hicimos de esta gramática, una gramática LL(1).

Más adelante corregimos ciertos elementos que eran innecesarios y de esta manera se simplificaba el código.

Por último, se creó una clase llamada *Síntesis*, en la que se encontraban todos los métodos para generar el String con el código HTML y por medio de llamadas desde el “.g4”, se iba traduciendo el código por medio atributos sintetizados.



Commit v0.16 (Casos de prueba actualizados v2)	origin & master	Rackumi	Yesterday 14:01
Commit v0.15 (Casos de prueba y lexico actualizados)		Rackumi	Yesterday 13:05
Merge remote-tracking branch 'origin/master'		Rackumi	Yesterday 12:15
Commit v0.14 (Introduccion de gestion de errores)		Rackumi	Yesterday 12:15
Commit v0.13 (Añadidos casos de prueba)		gocruz17	Yesterday 12:11
Commit v0.12 (Factorizacion y recursividad arreglados)		Rackumi	20/04/2021 20:23
Commit v0.11 (Limpieza de codigo v3 Factorizacion)		Rackumi	20/04/2021 20:01
Commit v0.10 (Limpieza de codigo v2 Recursividad)		Rackumi	20/04/2021 15:02
Commit v0.9 (Limpieza de codigo)		Rackumi	20/04/2021 14:46
Commit v0.9 (Arreglado error recursividad)		gocruz17	18/04/2021 18:32
Commit v0.8 (Hechas algunas factorizaciones)		Rackumi	18/04/2021 18:24
Commit v0.7 (Arreglado error recursividad)		gocruz17	18/04/2021 17:38
Commit v0.6 (Primera prueba sin errores)		gocruz17	18/04/2021 16:31
Commit v0.5 (prioridad mejorada y actualizacion del recognizer)		gocruz17	17/04/2021 17:36
Commit v0.4 (fin CAB, SIG, DIR)		gocruz17	13/04/2021 19:09
Commit v0.3 (inicio CAB, SIG, DIR)		Rackumi	13/04/2021 18:19
Commit v0.2 (recursividad izquierda arreglada en 4 sitios)		Rackumi	12/04/2021 21:54
Commit v0.1 (parte sintáctica opcional añadida)		Rackumi	12/04/2021 21:28
Initial Commit		Rackumi	08/03/2021 14:38

Traducción dirigida por sintaxis:

En cuanto a la traducción dirigida por sintaxis, se ha hecho únicamente uso de atributos sintetizados, de tipo String, en los que se ha ido introduciendo el código HTML y se ha ido concatenando.

Nivel Aprobado:

Para empezar, tendremos los 2 métodos principales del programa ya que son la estructura sobre la que se añadirán todos los demás. El primero será el método *inic()*, que creará la cabecera del HTML y mostrará el título del caso de prueba con el que se haya ejecutado el programa y abrirá la etiqueta del body para poder llamar a todos los demás métodos, y el método *end()*, que cerrará el body y el propio HTML para indicar la finalización de la escritura en el documento.

A continuación, tenemos el método *cabecera()*, que se encargará de como su propio nombre indica, mostrar las cabeceras de las funciones y procedimientos que se encuentren en el documento, enlazándolas con la propia función. Y por supuesto también se mostrarán las funciones y los procedimientos de forma completa gracias a distintas funciones menores concatenadas por atributos sintetizados.

También se han utilizado 3 métodos distintos para los tres estilos especiales mencionados en el enunciado, que son: *cte()*, *palres()* e *ident()*.

Nivel Opcional:

En cuanto a la parte opcional, tendremos la función *div()*, que se encargará de añadir un nivel de indentación al código que se encuentra entre un “inicio” y un “fin”. Tiene un correcto funcionamiento en cuanto a la anidación.

Para la comprobación de si existe una función o un procedimiento llamado “Principal”, tendremos una función llamada *esPrincipal()*, que se encargará de eso y por último para enlazar las variables o llamadas a funciones o procedimientos, se hará uso de las funciones *referencia()* y *referenciado()*.

Casos de prueba:

Pasa estos 8 casos de prueba, estarán separados por la mitad en la que una mitad estará dedicada a los casos de correcta estructura y funcionamiento, y los otros cuatro, estarán dedicados a mostrar los errores.

Cada caso estará tematizado para mostrar las distintas funcionalidades de nuestro programa.

Hay un noveno caso de prueba, en el que comprobará el error de 2 o más métodos llamados “Principal”.

Caso1A:

Este primer caso de prueba estará dedicado a la declaración de variables y el correcto funcionamiento del método Principal.

```
funcion entero nombre1()
    inicio
        a = +123;
        b = -69;
        c = 45;
        d = $+123;
        e = $-A6F9;
        f = $FFFF;
        g = +123.456;
        h = -0.69;
        i = 45.0;
        j = $+123.0;
        k = $-E.A6F9;
        l = $0.FFFF;
        caracter var5;
        entero var3;
        var4 = +56.65;
        var4 = -66335.3262;
        var4 += 23545;
        _var4 = +56.65;
        acumulador_total_2 -= +56.65;
        var5 = "Hola";
        varp1 = 'comilla doble " dentro';
        varp2 = "comilla simple ' dentro";
        varp3 = 'comilla simple '' dentro';
        varp4 = "comilla doble "" dentro";
        varp5 = 'comilla doble " y simple '' dentro';
        varp6 = "comilla simple ' y doble "" dentro";
        VAT = "?|`";
    fin

funcion entero Principal(entero var5, real var6)
    inicio
        entero var3;
        real var4;
        var3 /= var5;
        var4 = $0.FFFF;
        var5 = "Hola" ";
        var5 *= 'Maunuel" ';
        var5 = "Paco  " ";
    fin
```

Caso1B:

Este caso de prueba consistirá en los errores partiendo del caso 1A.

```
funcion entero nombre1()
    inicio
        a = +123;
        b = -6.9;
        c = 45;
        d = $+123;
        e = $-A6F9;
        f = $FFFF;
        g = +123.456;
        h = -0.69;
        i = 45.0;
        j = $+123.0;
        k = $-E.A6F9;
        l = $0.FFFF;
        caracter var5;
        entero var3;
        var4 = +56.65;
        var4 = -66335.3262;
        var4 += 23545;
        _var4 = +56.65;
        acumulador_total_2 -= +56.65;
        var5 = "Hola";
        varp1 = 'comilla doble " dentro';
        varp2 = "comilla simple ' dentro";
        varp3 = 'comilla simple dentro';
        varp4 = "comilla doble "" dentro";
        varp5 = 'comilla doble " y simple '' dentro';
        varp6 = "comilla simple ' y doble "" dentro";
        VAT = "?|`";
    fin

funcion entero nombre1(entero var5, real var6)
    inicio
        entero var3;
        real var4;
        var3 /= var5;
        var4 = $0.FFFF;
        var5 = "Hola' ";
        var5 *= 'Maunuel" ';
        var5 = "Paco ' ";
    fin
```


Caso2A:

En este segundo caso de prueba, se mostrarán ejemplos de operaciones y asignaciones.

```
funcion entero nombre1()
  inicio
    contador = var1 + var2;
    contador1 = ((4/1)/(4*6)) + 4;
    acumulador_total_2 += contador1 + contador;
    _contador3 = -1;
    var5 = (-66335.3262 * 33445);
    var4 += 23545;
    var6 = +56.65;
    var2 = "Hola" + 11;
  fin
funcion entero nombre1(entero var5, real var6)
  inicio
    var4 = (5252 * 3726) / 7822;
    var += var2;
    var /= $+0.FFFF;
    var = $-AAAA + $-A6F9;
    var *= $A.AFED;
  fin
```

Caso2B:

Este caso de prueba consistirá en los errores partiendo del caso 2A.

```
funcion entero nombre1()
  inicio
    contador = var1 + var2;
    contador1 = ((4/1)(4*6)) + 4;
    acumulador_total_2 += contador1 + contador;
    _contador3 = -1;
    var5 = (-66335.3262 * 33445);
    var4 += 23545;
    var6 = +56.65;
    var2 = "Hola" + 11;
  fin
funcion entero nombre1(var5, real var6)
  inicio
    var4 = (5252 * 3726) / 7822;
    var += var2;
    var /= $+0.FFFF;
    var = $-AAAA + $-A6F9;
    var *= $A.AFED;
  fin
```

Diagrama de anotaciones de errores:

- Señalando los operadores `(+, -, /, *)` en la línea `contador1 = ((4/1)(4*6)) + 4;`.
- Señalando el operador `+` en la línea `var4 += 23545;`.
- Señalando el operador `Identificador` en la línea `var2 = "Hola" + 11;`.
- Señalando el operador `Identificador` en la línea `funcion entero nombre1(var5, real var6)`.
- Señalando el operador `/ =` en la línea `var /= $+0.FFFF;`.

Caso3A:

Este caso de prueba consistirá en mostrar las distintas opciones del sent.

```
funcion entero nombre1()
    inicio
        bifurcacion (a == 5)
            entonces
                inicio
                    b = a + 23;
                    bucle
                        inicio
                            a = (b + 2) / c;
                            b = nombre1();
                            return a;
                        fin
                    hasta(cierto)
                fin
            sino
                inicio
                    return b;
                fin
        fin
funcion entero nombre2(entero var5, real var6)
    inicio
        entero var1;
        buclepara (i = 0; i <= var6; j = -2)
            inicio
                nombre2(var1);
                buclemientras (falso)
                    inicio
                        return -1;
                    fin
            fin
        return 5;
    fin
```

Caso3B:

Este caso de prueba consistirá en los errores partiendo del caso 3A.

```
funcion entero nombre1()
  inicio
    bifurcacion (a == 5);
    entonces
      inicio
        b = a + 23;
        bucle hasta(cierto)
          inicio
            a = (b + 2) / c;
            return a;
          fin
      fin
    sino
      inicio
        return b;
      fin
  fin
funcion entero nombre1(entero var5, real var6)
  inicio
    buclepara (i = 0 i <= var6; j = -2)
      inicio
        nombre1(var5, var6);
        bucle mientras (falso)
          inicio
            return -1;
          fin
      fin
  return 5;
fin
```

Caso4A:

En este último caso de prueba, se dedicará a mostrar los comentarios.

```
procedimiento nombre1(entero variable1, entero variable2)
    inicio
        return 0;
    fin

procedimiento nombre1()
    inicio
        return 0;
    fin

%% Hola esto es un comentario de una linea

funcion entero nombre1()
    inicio
        return 0; %% Comentario despues de un elemento
    fin

funcion entero nombre1()
    inicio
        return 0; %% Comentario despues de un elemento
    fin
%- Hola esto es un comentario
multilinea -%
```

Caso4B:

Este caso de prueba consistirá en los errores partiendo del caso 4A.

```
procedimiento nombre1(entero variable1, entero variable2)
    inicio
        return 0;
    fin

% Hola esto es un comentario de una linea
↑

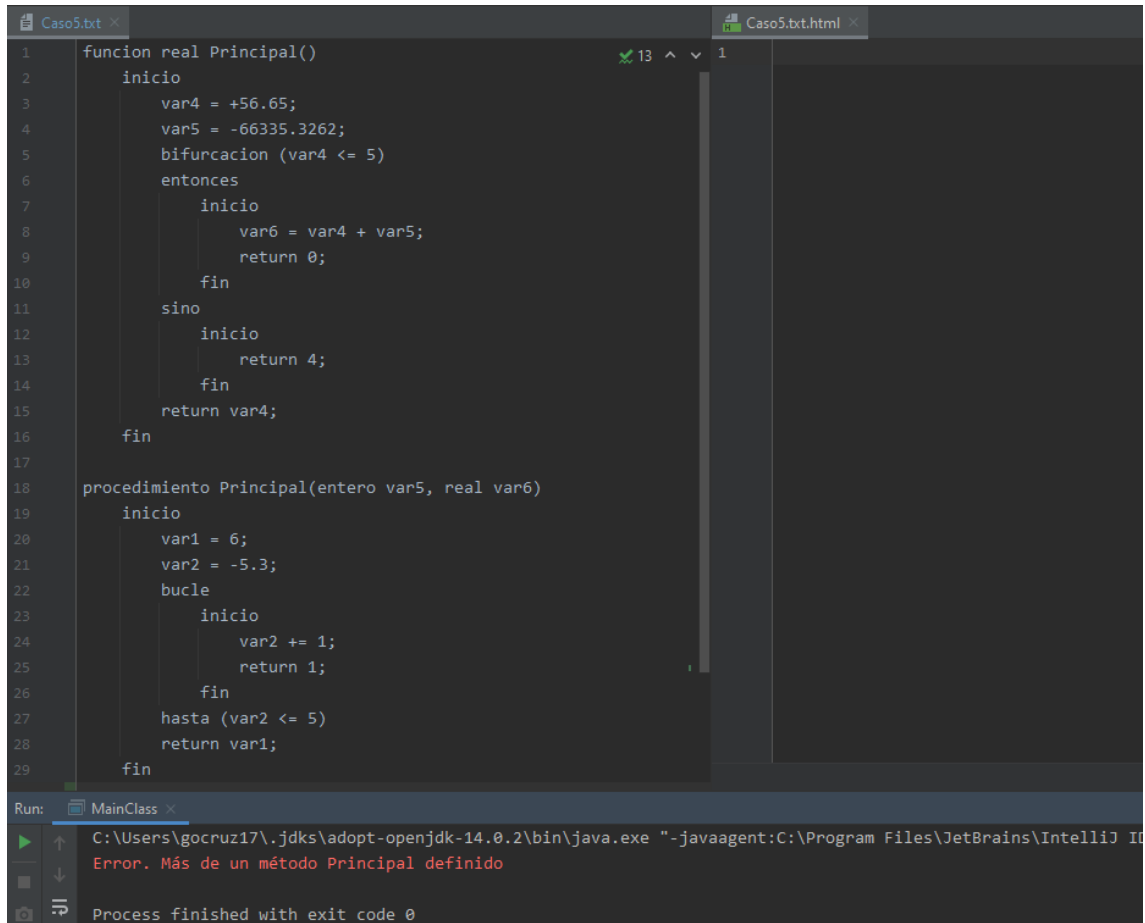
funcion entero nombre1()
    inicio
        return 0; %- Comentario despues de un elemento
    fin hola-%

%- Hola esto es un comentario
multilinea-%
↑
inicio
fin
```

Caso5:

Este caso de prueba consistirá en mostrar el error generado por dos o más funciones o métodos llamados Principal.

En este caso, a diferencia los demás casos de error, generará el fichero .html pero vacío, ya en el enunciado lo hemos interpretado de esta manera.



```
1  funcion real Principal()  
2      inicio  
3          var4 = +56.65;  
4          var5 = -66335.3262;  
5          bifurcacion (var4 <= 5)  
6              entonces  
7                  inicio  
8                      var6 = var4 + var5;  
9                      return 0;  
10                 fin  
11             sino  
12                 inicio  
13                     return 4;  
14                 fin  
15             return var4;  
16         fin  
17  
18     procedimiento Principal(entero var5, real var6)  
19         inicio  
20             var1 = 6;  
21             var2 = -5.3;  
22             bucle  
23                 inicio  
24                     var2 += 1;  
25                     return 1;  
26                 fin  
27             hasta (var2 <= 5)  
28             return var1;  
29         fin
```

Run: MainClass x

C:\Users\gocruz17\.jdk\adopt-openjdk-14.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ I
Error. Más de un método Principal definido

Process finished with exit code 0

Anexo:

Cabeceras:

```
CAB(program) = CAB'(part) U CAB'(part) = {"function"}

CAB(part) = {"function", "procedimiento"}

CAB(restpart) = {IDENTIFICADOR}

CAB(listparam) = CAB'(listparam) U CAB'(type) = {"entero", "real", "caracter"}

CAB(type) = {"entero", "real", "caracter"}

CAB(blq) = {"inicio"}

CAB(sentlist) = CAB'(sentlist) U CAB'(sent) =
  {"entero", "real", "caracter", "inicio", "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}

CAB(sent) = CAB'(type) U CAB'(blq) U {IDENTIFICADOR} U {"return"} U {"bifurcacion"} U
{"buclepara"} U {"buclemientras"} U {"bucle"} =
  = {"entero", "real", "caracter", "inicio", "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}

CAB(lid) = {IDENTIFICADOR}

CAB(asig) = {"="} U {"+="} U {"-="} U {"*="} U {"!="} =
  = {"=", "+=", "-=", "*=", "!="}

CAB(exp) = CAB'(exp) U {IDENTIFICADOR} U {"("} U {CONSTENTERO} U {CONSTREAL} U
{CONSTLIT} =
  {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}

CAB(op) = {"+"} U {"-"} U {"*"} U {"/" } =
  = {"+", "-", "*", "/"}

CAB(lcond) = CAB'(lcond) U CAB'(cond) U {"no"} =
  = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT} U {"no"} =
  = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT, "no"}

CAB(cond) = CAB'(exp) U {"cierto"} U {"falso"} = {IDENTIFICADOR} U {"("} U
{CONSTENTERO} U {CONSTREAL} U {CONSTLIT} =
  = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}

CAB(opl) = {"y"} U {"o"} = {"y", "o"}

CAB(opr) = {"=="} U {"<>"} U {"<"} U {">"} U {">="} U {"<="} =
  = {"==", "<>", "<", ">", ">=", "<="}
```

Siguientes:

En cuanto a los siguientes, en este caso, no han hecho falta calcularlos, ya que, para la creación de los directores, al no haber λ , no serán necesarios.

Directores:

Estos directores están hechos antes de corregir la recursividad por la izquierda, para resolver la factorización cuando había más de un símbolo de anticipación. También deberíamos haber hecho los directores de las partes en las que se ha arreglado la recursividad, pero no hizo falta ya que eran visibles de forma clara.

```
DIR(program ::= part program) = CAB'(part) = {"function", "procedimiento"}
DIR(program ::= part) = CAB'(part) = {"function", "procedimiento"}

DIR(part ::= "funcion" type restpart) = {function}
DIR(part ::= "procedimiento" restpart) = {procedimiento}

DIR(restpart ::= IDENTIFICADOR "(" listparam ")" blq) = {IDENTIFICADOR}
DIR(restpart ::= IDENTIFICADOR "(" " " ")" blq) = {IDENTIFICADOR}

DIR(listparam ::= listparam "," type IDENTIFICADOR) = {"entero", "real", "caracter"}
DIR(listparam ::= type IDENTIFICADOR) = {"entero", "real", "caracter"}

DIR(type ::= "entero") = {"entero"}
DIR(type ::= "real") = {"real"}
DIR(type ::= "caracter") = {"caracter"}

DIR(blq ::= "inicio" sentlist "fin") = {"inicio"}

DIR(sentlist ::= sentlist sent) = {"entero", "real", "caracter", "inicio", "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}
DIR(sentlist ::= sent) = {"entero", "real", "caracter", "inicio", "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}

DIR(sent ::= type lid ";") = {"entero", "real", "caracter"}
DIR(sent ::= IDENTIFICADOR asig exp ";") = {IDENTIFICADOR}
DIR(sent ::= "return" exp ";") = {"return"}
DIR(sent ::= IDENTIFICADOR "(" lid ")" ";" ) = {IDENTIFICADOR}
DIR(sent ::= IDENTIFICADOR "(" " " ")" ";" ) = {IDENTIFICADOR}
DIR(sent ::= "bifurcacion" "(" lcond ")" "entonces" blq "sino" blq) = {"bifurcacion"}
DIR(sent ::= "buclepara" "(" IDENTIFICADOR asig exp ";" lcond ";" IDENTIFICADOR asig exp ")" blq) = {"buclepara"}
DIR(sent ::= "buclemientras" "(" lcond ")" blq) = {"buclemientras"}
DIR(sent ::= "bucle" blq "hasta" "(" lcond ")") = {"bucle"}
DIR(sent ::= blq) = {"inicio"}

DIR(lid ::= IDENTIFICADOR) = {IDENTIFICADOR}
DIR(lid ::= IDENTIFICADOR "," lid) = {IDENTIFICADOR}

DIR(asig ::= "=") = {"="}
DIR(asig ::= "+=") = {"+="}
DIR(asig ::= "-=") = {"-="}
DIR(asig ::= "*=") = {"*="}
DIR(asig ::= "/=") = {"/="}

DIR(exp ::= exp op exp) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}
DIR(exp ::= IDENTIFICADOR "(" lid ")") = {IDENTIFICADOR}
DIR(exp ::= "(" exp ")") = {"("}
DIR(exp ::= IDENTIFICADOR) = {IDENTIFICADOR}
DIR(exp ::= CONSTENTERO) = {CONSTENTERO}
DIR(exp ::= CONSTREAL) = {CONSTREAL}
DIR(exp ::= CONSTLIT) = {CONSTLIT}
```



```
DIR(op ::= "+") = {"+"}
DIR(op ::= "-") = {"-"}
DIR(op ::= "*") = {"*"}
DIR(op ::= "/" ) = {"/"}
```

```
DIR(lcond ::= lcond opl lcond) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL,
CONSTLIT, "no"}
DIR(lcond ::= lcond) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT, "no"}
DIR(lcond ::= "no" cond) = {"no"}
```

```
DIR(cond ::= exp opr exp) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}
DIR(cond ::= "cierto") = {"cierto"}
DIR(cond ::= "falso") = {"falso"}
```

```
DIR(opl ::= "y") = {"y"}
DIR(opl ::= "o") = {"o"}
```

```
DIR(opr ::= "==") = {"=="}
DIR(opr ::= "<>") = {"<>"}
DIR(opr ::= "<") = {"<"}
DIR(opr ::= ">") = {">"}
DIR(opr ::= ">=") = {">="}
DIR(opr ::= "<=") = {"<="}
```